12 La shell (2)

Stiamo esplorando le potenzialità della shell, uno dei programmi di base per Linux, che ci permette di capire alcuni concetti fondamentali. Nella lezione precedente abbiamo spiegato a grandi linee alcuuni concetti relativi al file system.
Un'altra componente fondamentale di Linux è la gestione dei **processi**. Ogni programma che viene eseguito genera almeno un processo. Questo può essere visto come un'entità che compie determinate operazioni. Ad esempio, la shell stessa è un processo che ci permette di digitare comandi e che li esegue. Mozilla è un processo (più d'uno, a dire la verità) che ci permette di navigare in Internet, e così via. Quando con il mouse facciamo partire dei programmi, non stiamo facendo altro che mandare in esecuzione più processi, che condivideranno il processore e le altre risorse (si è accennato a ciò nella prima lezione del corso).
Quando digitiamo un comando da shell, stiamo mandando in esecuzione un programma, composto da uno o più processi. Ma abbiamo visto che, una volta digitato un comando (per esempio xeyes), la shell si blocca fino alla terminazione del programma stesso. Possiamo mandare in esecuzione più programmi (quasi) nello stesso istante mettendo un carattere **&** dopo il loro nome. Ad esempio, digitando "xeyes &", si aprirà una finestra come prima, ma la shell non si bloccherà. Dopo averci restituito due numeri, è tornata ad accettare comandi. Possiamo a questo punto eseguire un altro comando, se vogliamo (provate ad esempio ad eseguire un altro "xeyes &",. e poi un "ls").
I due numeri che vengono stampati per ogni programma hanno, ovviamente, un senso. Il primo è l'identificativo di job, cioè il modo in cui la shell da cui è stato lanciato chiama un programma. Con il comando "jobs", è possibile vedere la lista dei programmi attivi lanciati dalla shell, unitamente al loro stato (nel nostro caso è "running".
Il secondo valore è invece l'identificativo di processo (PID), con cui Linux chiama il processo stesso (e non è relativo a nessuna shell). Con questi due valori possiamo manipolare i processi in tanti modi, ricordando che il primo ha valore solo dalla stessa shell da cui è stato lanciato un processo, mentre il secondo ha valore globale. Se vogliamo uccidere un processo che per esempio non risponde alla chiusura via mouse, possiamo scrivere "kill -9 PID". Nel mio caso, il secondo comando di xeyes ha restituito i valori "[2] 1134". Se scrivo "kill -9 1134", il programma effettivamente muore (provate anche voi sostituendo a 1134 il valore che vi è stato restituito). Si può anche scrivere "xkill", e quando il puntatore del mouse cambia forma, fare click sopra la finestra che vogliamo chiudere (attenti a non sbagliare finestra o a non far partire un click involontario!). Questo secondo metodo, tuttavia, è di solito meno efficace del primo.

Che fare se lanciamo un programma dimenticandoci il simbolo '&' alla fine? E' sufficiente premere [CTRL][z] sulla shell per sospenderlo. Provate a lanciare un ulteriore "xeyes" senza '&', e sospendetelo. Notate che ci viene restituito un numero solo (l'identificativo di job), e che il

programma entra in stato di "STOPPED": infatti non seguirà il vostro mouse con gli occhietti. Per farlo ripartire, liberando però la shell, è sufficiente digitare "bg identificativo_di_JOB", per esempio nel mio caso "bg 2". Questo signica mandare in **background** un processo. Si può fare l'opposto con "fg identificativo_di_JOB" (nel mio caso "fg 2"), e la shell smetterà di accettare comandi per attendere la terminazione del comando che abbiamo appena mandato in **foreground**.
Ci sono dei casi in cui vogliamo che due o più comandi partano uno dopo l'altro.
E' sufficiente scrivere la sequenza dei comandi separati dal carattere '**;**' (ad esempio "comando1 ; comando2; comando3"), perché ciò avvenga. Come esempio, supponiamo di voler fare un "ls", seguito dopo cinque secondi da un "xeyes" (odierete questo programmino, ne sono certo!). Scrivete dunque "ls ; sleep 5 ; xeyes", dove sleep è un comando che accetta un numero, e dorme per il numero di secondi indicato: vi apparirà il contenuto della vostra home, quindi non succederà niente per cinque secondi, e degli ulteriori occhietti malefici popoleranno il vostro schermo.

Per vedere che processi sono stati lanciati, esiste il comando "ps". Se gli si passa il parametro -a (digitando quindi "ps -a"), sarà possibile vedere tutti i processi che sono in esecuzione sul vostro computer con associato il vostro nome utente. Potete uccidere un qualsiasi processo con il comando "kill -9 PID", ma non vi consiglio di farlo se non sapete cosa morirà! Se siete curiosi di sapere il nome di tutti i processi che girano sul vostro computer, e non solo quelli relativi al vostro nome utente, potete digitare "ps -aux". Di solito la lista di processi che fanno riferimento a root è molto lunga: questi sono i programmi di sistema, che gestiscono i terminali, la rete, etc.

E' anche possibile che l'output testuale restituito da un programma sia dato come input ad un altro programma, anche se questa è un'opzione per utenti un po' più esperti. Basterà mettere tra i nomi dei due comandi il carattere '|' (chiamato **pipe**). Per esempio, supponiamo di volere il listato delle directory contenute in / in ordine alfabetico inverso. Il lilstato viene restituito dal comando "ls /", mentre esiste un comando di nome sort che, se invocato con "sort -r" che ordina liste di parole (per esempio, scrivete sulla shell "sort -r" seguito da [Invio], quindi scrivete delle parole dando [Invio] tra una e l'altra, e premete [CTRL][d] dopo l'ultima). Il nostro comando composto è dunque "ls / | sort -r". L'effetto dovrebbe essere quello voluto!

Se volete che l'output testuale di un comando venga scritto su un file invece che a video, dovete scrivere nel seguente modo: "comando > nome_del_file". Per esempio, scrivendo "ls > listato", creerete un file di nome "listato", al cui interno è contenuto il listato della directory in cui si trova. Per vedere il contenuto di un file, è sufficiente digitare "more nome_del_file", quindi nel nostro caso "more listato". Se si ridirige l'output su un file già esistente, questo viene cancellato e riscritto. Se si vuole evitare questo evento, bisogna usare '>>' invece di '>'. In queso caso l'output del comando viene messo dopo il contenuto del file, se questo esiste già (provate con "ps >> listato" seguito da "more listato").

Uno strumento di enorme utilità è il **pattern marching**. Per esempio, abbiamo visto più volte che il comando ls restituisce la lista di files e directory contenuti in una certa directory. Supponiamo di essere interessati a tutti i files contenuti il cui nome inizia per m. Dobbiamo digitare "ls -d /m*" per fare quanto voluto (il parametro -d dice ad ls di non mostrare il contenuto delle directory, ma solo il loro nome). Il carattere '*' viene interpretato dalla shell come un jolly, che può essere sostituito da qualsiasi parola o sequenza di caratteri. La voce m* significa dunque: qualsiasi sequenza di caratteri che inizia per m. Allo stesso modo *r significa qualsiasi sequenza di caratteri che terminano per r (provate con "ls -d /*r"). E' possibile anche avere più caratteri *, come in m*n* (cosa significa?).
Il carattere ?, invece viene interpretato come un carattere singolo qualsiasi. Dunque m? significa qualsiasi sequenza di due caratteri inizianti per m. Fate una prova con "ls -d /u?r".

Un comando fondamentale di Linux è "man", che avvia il manuale. Il sistema operativo e tutti i suoi comandi sono spiegati in dettaglio in numerossime pagine di manuale (probabilmente verranno installate quelle in inglese, ma ne esiste anche una versione in italiano che presto impareremo ad usare). Per esempio potete digitare "man ls", per vedere quanto un comando semplice come ls possa in realtà essere complesso da conoscere nella sua interezza. Potete sbizzarrirvi anche con "man xeyes"!

Tutte le cose che sono state esposte in questa lezione sono ovviamente combinabili tra di loro. E' per esempio possibile digitare il comando "(sleep 20 ; ls > listato) &" per mandare in background una sequenza di comandi il cui secondo scrive su un file. Non esistono limiti alla complessità dei comandi che si possono eseguire (e nemmeno alla perversione degli utenti!).

Nonostante a primo acchito possa sembrare molto complicato usare la shell, oltre che perfettamente sostituibile dall'interfaccia grafica, questo non è sempre vero.
Immaginate per esempio che la vostra interfaccia grafica si blocchi per qualche motivo (mouse che non si muove, tutto bloccato). E' un evento piuttosto raro, a dire la verità, ma può succedere. Se il blocco non è a livello del sistema (e dunque talmente grave da non lasciare altra opzione che il riavvio), è possibile aprire una shell testuale ([CTRL][ALT][F1]), e controllare cosa non va. Si può quindi far ripartire l'interfaccia grafica in pochi secondi, e tornare al lavoro!
Ci sono numerosi altri casi in cui la shell è utile, ma questi si apprendono con l'esperienza. Quindi non posso che consigliarvi di continuare ad usarla, malgrado la sua versione grafica (che di sicuro è un po' più lenta). Nella prossima esercitazione avrete modo di allenarvi con quanto presentato in queste lezioni, e anche su qulcosa in più!